Apple IIGS
#35: Printer Driver Specifications

Revised by: Matt Deatherage September 1990
Written by: Dan Hitchens, Matt Deatherage & Suki Lee May 1988

This Technical Note describes the routines and internal structures needed to design a printer driver for the Apple IIGS system, and you should use this Note with the Apple IIGS Toolbox Reference manuals. An overview and associated parameters for each of the printer driver routines are in the Print Manager chapter, and you should refer to these for a complete picture.
Changed since March 1990: Added corrections and further descriptions.

Printing Modes

There are two printing modes: immediate and deferred.

- In **immediate mode**, pages are printed as they are drawn into the printing `grafPort`. As the application makes QuickDraw II calls, the printer driver immediately generates commands, transferring ink to page when the page is closed. This is the fastest form of printing, but only produces high-quality images on printers that can translate QuickDraw II commands to other graphic commands. For example, the LaserWriter driver translates the QuickDraw II calls into PostScript® calls which can produce high-quality images.

- In **deferred mode** (sometimes referred to as **spool mode**), pages are captured to memory or disk and printed after all pages have been defined. Most printer drivers use deferred mode to create high-quality images. Since most drivers cannot obtain enough memory to image an entire page at once, they redraw page in several pieces, or **bands**. The printer driver creates a `grafPort` whose `boundsRect`, `portRect`, `clipRgn`, and `visRgn` correspond to the band and plays the picture back, thus causing the saved commands to draw only the images which fall within the band. Once the pixel image for the band is created, the printer driver converts the image to printer codes and sends the codes to the printer through the port driver.

File Structure

The user can install new printer drivers into the system by copying a printer driver file into a subdirectory called DRIVERS within the SYSTEM subdirectory. The printer driver file must be of type $BB and have an auxiliary type of $0001.

Print Driver Calls

A printer driver must support the following calls:

`PrDefault`	$0913	Sets print record to default
`PrValidate`	$0A13	Validates print record
`PrStlDialog`	$0B13	Performs a style dialog
`PrJobDialog`	$0C13	Performs a job dialog
`PrPixelMap`	$0D13	Prints a pixel map
`PrOpenDoc`	$0E13	Opens the document
`PrCloseDoc`	$0F13	Closes the document
`PrOpenPage`	$1013	Opens a page
`PrClosePage`	$1113	Closes a page
`PrPicFile`	$1213	Prints a picture file
--RESERVED--	$1313	
`PrError`	$1413	Gets the error value
`PrSetError`	$1513	Sets the error value
`GetDeviceName`	$1713	Gets device's name
`PrDriverVer`	$2313	Gets installed driver version

Printer drivers **may** support the following calls if they use the new driver structure outlined below:

`PrGetPrinterSpecs`	$1813	Returns printer type and characteristics
`PrGetPgOrientation`	$3813	Returns page orientation

Print Driver Entry

- For older drivers, entry is at the first byte (no offset). For newer (Print Manager 3.0 and later) drivers, the first word is $0000, indicating a new style driver. The next word is a count of how many calls this driver supports. All drivers **must** support the minimum call set. Additional calls must be supported in the sequence listed (for example, if a driver supports `PrGetPgOrientation`, it must also support `PrGetPrinterSpecs`).
- The Print Manager places an index to the correct routine in the `X` register (see the example and note the specific ordering of the routines) .
- There are two long return addresses (six bytes) that have been pushed onto the stack. (You must take these addresses into account to access the parameters and to return correctly.)

Example

```
StartOfNewDriver    START

                    dc i2 '0'                          ; new style driver
                    dc i2'(ListEnd-PrDriverList)/4'    ; count

                    jmp (PrDriverList,x)
```

```
PrDriverList        dc a4'PrDefault'
                    dc a4'PrValidate'
                    dc a4'PrStlDialog'
                    dc a4'PrJobDialog'
                    dc a4'PrDriverVer'
                    dc a4'PrOpenDoc'
                    dc a4'PrCloseDoc'
                    dc a4'PrOpenPage'
                    dc a4'PrClosePage'
                    dc a4'PrPicFile'
                    dc a4'InvalidRoutine'
                    dc a4'PrError'
                    dc a4'PrSetError'
                    dc a4'GetDeviceName'
                    dc a4'PrPixelMap'
                    dc a4'PrGetPrinterSpecs'
                    dc a4'PrGetPgOrientation'
ListEnd             anop
```

In previous versions of this Note, the `PrPixelMap` and `PrDriverVer` entries were reversed.

Note that when using the above technique, you're using a 16-bit jump into a table of 24-bit addresses. If all your entry points are in the same segment, this is not a problem.

If your routines' entry points are not all in the same segment, you need a dispatching routine like the following:

```
StartOfNewDriver    START

                    dc i2 '0'                           ; new style driver
                    dc i2 '(ListEnd-PrDriverList)/4'    ; count

                    lda PrDriverList+2,x
                    sep #$20
                    pha                                 ; push high byte of address
                    rep #$20
                    lda PrDriverList,x
                    dec a                               ; decrement low 2 bytes only
                    pha                                 ; push modified low word of
address
                    rtl                                 ; transfer to the routine
```

See Apple IIGS Technical Note #90, 65816 Tips and Pitfalls, for a discussion of dispatching with `RTL`.

Print Driver Exit

When one of your routines is ready to exit, it needs to remove the input parameters from the stack, leaving the result space (if any) and the two `RTL` addresses. Set the accumulator and the carry flag to reflect any error you are returning, then perform an `RTL`.

Example

If there are N bytes of input parameters to remove, use something like the following. This code assumes that the error code is in the accumulator.

```
                    tay                                 ; keep error code in Y
temporarily
                    lda 5,s
                    sta N+5,s
                    lda 3,s
                    sta N+3,s
                    lda 1,s
                    sta N+1,s
                    tsc
                    clc
                    adc #N
                    tcs
                    tya                                 ; get error code
                    cmp #1                              ; set carry if error is not
zero
                    rtl
```

Figure 1 diagrams the stack just before exiting the print driver:

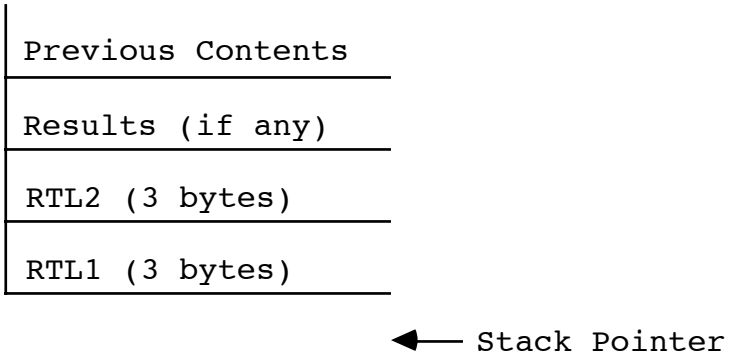

Figure 1–Stack Prior to Exiting the Print Driver

You should do an `RTL` with the contents of the flags and registers set appropriately. (See the Return from Call section of the "Using The Apple Tools" chapter of the *Apple IIGS Toolbox Reference*.)

Print Record Structure

Since application programs often need to fiddle with parts of the print record (i.e., the values in the style subrecord), we have defined ways for applications to interpret the print record, and specifically the style subrecord.

`iDev`, the first word of the printer information subrecord, has two defined values for third-party printer drivers. A value of $8001 indicates a dot-matrix printer while a value of $8003 indicates a laser printer.

A value of $8001 indicates that fields of the style subrecord should be interpreted as they are by the ImageWriter driver, as documented in the *Apple IIGS Toolbox Reference*. The first seven bits (0–6) of `wDev` are defined as for the ImageWriter driver. Bits 7-11 are reserved for Apple's use and must be set to zero. Bits 12-15 may be used by third-party printer drivers as necessary; these bits are set to zero in Apple's drivers.

A value of $8003 indicates that fields of the style subrecord should be interpreted as they are by the LaserWriter driver. The first four bits (0–3) of `wDev` are defined as for the LaserWriter driver. Bits 4-11 are reserved for Apple's use and must be set to zero. Bits 12-15 may be used by third-party printer drivers as necessary; these bits are set to zero in Apple's drivers.

If an application wishes to take advantages of specific features of a third-party printer driver, it has to know that it is dealing with that driver. Since all drivers look pretty much alike, the Print Manager allows you to ask for the name of the currently selected printer driver. An application may make the Print Manager call `PMGetPrinterName`, which is documented in Volume 3 of the *Toolbox Reference*. The Print Manager returns the name of the currently selected printer in a Pascal (length byte) string. The name returned is the name of the **file** from which the driver was loaded. If you intend to use this method to identify a driver, you must inform users **not** to rename the Printer Driver file on the boot disk.

For alternate driver identification, Developer Technical Support assigns new `iDev` values if you feel it is absolutely necessary for your driver. Please keep in mind, however, that no application knows how to interpret style records for non-standard `iDev` values, and that Apple does not publish such interpretations.

Print Driver Calls

Your printer driver handles the following calls:

PrDefault ($0913)

Description:
Fills the fields of the specified print record with default values for the printer.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record

Returned:
None

Performs the following:
- Validates that `PrintRecordHandle` is a handle and does nothing if not.
- Determines the default values for the print record either through tables or calculations. The default values should take into account such things as paper size and orientation, print mode, printer type, etc.

- Copies the default values to the print record specified by the `PrintRecordHandle` parameter.

PrValidate ($0A13)

Description:
Checks the print record to see that it is valid for the currently installed printer driver.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record

Returned:

`ChangeFlag`	WORD	Boolean; TRUE if the record is adjusted

Performs the following:
- Checks to see if the print record is from this particular driver.
- If the print record is not from this driver, it uses the default values for this driver.
- If the print record is from this driver, it makes any changes that might be needed (i.e., style, paper size, etc.).

PrStlDialog ($0B13)

Description:
Performs a style dialog with the user.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record

Returned:

`ConfirmFlag`	WORD	Boolean; TRUE if the dialog is confirmed

Performs the following:
- Conducts a style dialog with the user to determine the page dimensions and other information needed for page setup (the initial settings of the dialog are derived from the print record).
- If the user confirms the dialog, the information from the dialog is saved in the specified print record, `PrValidate` is called, and the routine returns TRUE.
- If the user cancels the dialog, the print record is left unchanged, and the routine returns FALSE.

Note: The following are items typically found in printer style dialogs:

- Paper Size (US Letter, US Legal, A4 Letter, B5 Letter, International Fanfold)
- Printing Orientation (Landscape, Portrait)
- Vertical Sizing (Normal, Intermediate, Condensed)
- Special Effects:
  - Font Effects (Font Substitution, Smoothing)
  - Reduction or Enlargement
  - Gaps or No Gaps between pages

Every printer style dialog should have an OK button (default) and a Cancel button.

Note: When calling other routines in your printer driver (like `PrValidate`), be sure to do so through the Tool Dispatcher ($E10000 or $E10004) so any necessary patches have an opportunity to execute.

PrJobDialog ($0C13)

Description:
Performs a job dialog with the user.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record

Returned:

`ConfirmFlag`	WORD	Boolean; True if the dialog is confirmed

Performs the following:
- Conducts a job dialog with the user to determine the print quality, range of pages to print, and other specifications. The initial settings are derived from the previous `PrJobDialog` call (or initial default values) except the page range which is set to ALL, and the number of copies which is set to ONE.
- If the user confirms the dialog, `PrValidate` is called, the print record is updated, and the routine returns TRUE.
- If the user cancels the dialog, the print record is left unchanged, and the routine returns FALSE.

Note: The following are items typically found in printer job dialogs:

- Print Quality (Best, Faster, Draft, etc.)
- Color option
- Pages (All, Range)
- Copies
- Paper Source (paper cassette, manual feed)

Every printer job dialog should have an OK button (default) and a Cancel button.

Note: When calling other routines in your printer driver (like `PrValidate`), be sure to do so through the Tool Dispatcher ($E10000 or $E10004) so any necessary patches have an opportunity to execute.

PrPixelMap ($0D13)

Description:
Prints all or part of the specified pixel map.

Passed:

`srcLocPtr`	LONG	Pointer to the source `LocInfo` which contains the pointer to the pixel map.
`srcRectPtr`	LONG	Pointer to the rectangle which encloses the pixel map to be printed.
`colorFlag`	WORD	Boolean; FALSE if black and white, TRUE if color.

Returned:
None
Performs the following:
- Calls `DevIsItSafe` (port driver call) to verify that the port it functioning and it is safe to proceed. If it is not functioning, set the internal error code to $1302 (`Port Not On`) and return with an error status.
- Saves the current `grafPort`.
- Turns on the watch cursor to signal the user that it will take some time.
- Clears the internal error code (default, if no errors occur).

You can choose to print the pixel map in any convenient fashion; one convenient way is to allocate a new print record and call your normal printing routines. This method is outlined below.

- Gets a new handle for a print record and set it to the defaults by calling `PrDefault`.
- If `colorFlag` is set, change the style subrecord of the print record to reflect color printing.
- Do any initialization that might be needed by the driver.
- Determine the intersection of the two rectangles (rectangle pointed to by `srcRectPtr` and the pixel map's boundary rectangle from `srcLocPtr`) and if there is no intersection, then nothing is to be printed.
- Print the pixel image which is within the intersection of the two rectangles.
- Cause a page eject to occur on the printer.
- Do any clean up that is needed.
- Turn off the watch cursor by calling `InitCursor` (or restore the previous cursor using `SetCursor`).
- Restore the `grafPort` by calling `SetPort`.

PrOpenDoc ($0E13)

Description:
This routine initializes the things needed to open a document. In deferred mode, it establishes a `grafPort` and makes it the current port for printing.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record
`PrinterPortPtr`	LONG	Pointer to the `grafPort`, if desired, zero to allocate a new `grafPort`

Returned:

`PrinterPortPtrRet`	LONG	Pointer to the `grafPort` if the `PrinterPortPtr` was zero

Performs the following:
- Calls `DevIsItSafe` (port driver call) to verify that the port is functioning and it is safe to proceed.
- Turns on the watch cursor to signal the user that it will take some time.
- Validates the print record passed by calling `PrValidate`.

- Clears the internal error code (default, if nothing happens).
- Puts up a dialog indicating that printing is occurring (or preparing to print).
- If the user needs a `grafPort`, create one and internally note that one was created (`PrCloseDoc` needs to know that one was created here).
- Initializes parameters (i.e., page number, document number, etc.).
- If deferred mode, create an initial page list (an array of handles to pictures) for recording pages. You can pick an arbitrary number to start with (like 20). This assumes spooling to memory; spooling to disk will obviously be different.
- Do other initialization that might be needed to start a print job.

Possible errors:

`portNotOn`	$1302	Indicates Port Not On

PrCloseDoc ($0F13)

Description:

Closes the `grafPort` being used for printing. For immediate mode, this routine ends the printing job. For deferred mode, this routine ends the recording of the document to be printed.

Passed:

`PrintGrafPortPtr`	LONG	Pointer to the `grafPort` used for printing

Returned:

None

Performs the following:

- Checks that the last print driver call did not cause a Port Not On error. If the error occurred, do nothing and return.
- Call `ClosePort` to close the printing `grafPort`.
- If the driver allocated a `grafPort` in `PrOpenDoc`, disposes of it.
- If in immediate mode, does what is needed to shut things down.
- Takes down the information dialog box from `PrOpenDoc`.

Possible errors:

`portNotOn`	$1302	Indicates Port Not On
`prBozo`	$13FF	Someone unloaded the driver in the middle of the print loop

PrOpenPage ($1013)

Description:

Begins a new page only if the page falls within the page range specified in the job subrecord.

Passed:

`PrintGrafPortPtr`	LONG	Pointer to the `grafPort` used for printing
`PageFramePtr`	LONG	Pointer to the `scaling` parameter, zero for none.

Returned:

None

Performs the following:

- Looks at the driver's internal error value, and if an error has occurred, it returns without doing anything.
- Increments the page number.
- Calls `SetPort` to make the specified port the current port.
- Initializes the port and zeroes the boundary rectangle so no actual drawing occurs.
- If immediate mode, then do the following:

  If this page is to be printed, install immediate mode procedures by doing the following:

- Create a procedure table (get the standard procedures from `SetStdProcs`).
- Put pointers to your procedures into the table and call the QuickDraw II routine `SetGrafProcs`. This causes QuickDraw II calls to call your routines instead of drawing to the pixel map associated with the `grafPort`.

- If deferred mode, then do the following:
    - If the current page is out of the page range, then return without doing anything further.
    - If the user passes his own `PageFramePtr` , then get it.
    - Open a picture by calling `OpenPicture` and adding its handle to the page list array described in `PrOpenDoc`.
    - Set the `ClipRgn` and `VisRgn` to the sizing framing rectangle specified by PageFramePtr , or if none was specified, to the default of `rPage`.

Possible errors:

`portNotOn`	$1302	Indicates Port Not On
`prBozo`	$13FF	Someone unloaded the driver in the middle of the print loop

PrClosePage ($1113)

Description:
This signals the end of a page.

Passed:

`PrintGrafPortPtr`	LONG	Pointer to the `grafPort` used for printing

Returned:
None

Performs the following:
- Looks at the driver's internal error value and if a Port Not On error has occurred, it returns without doing anything.
- If immediate mode, do the following:
    - If the current page is within the range of pages to be printed, then cause a form feed (unless no gap was specified).
- If deferred mode, do the following:
    - If there was no picture generated, then do nothing and just return.
    - Restore the `grafPort` to the port saved in `PrOpenPage`.
    - Do a `ClosePicture` to close the picture.

Possible errors:

`portNotOn`	$1302	Indicates Port Not On
`prBozo`	$13FF	Someone unloaded the driver in the middle of the print loop

PrPicFile ($1213)

Description:
Prints a picture file generated in deferred mode.

Passed:

`PrintRecordHandle`	LONG	Handle to the print record
`PrintGrafPortPtr`	LONG	Pointer to the `grafPort` used for printing
`StatusRecPtr`	LONG	Pointer to the printer status record

Returned:
None

Performs the following:
- Looks at the driver's internal error value and if a Port Not On error has occurred, it returns without doing anything.
- If immediate mode, return without doing anything.
- If deferred mode, then do the following:
  - If the error code is not zero (errors) then dispose of all the recorded page images.
  - Put up an information dialog indicating that printing is occurring.
  - Display a watch cursor (saving the current cursor first if you like).
  - If `PrintGrafPortPtr` is `NIL`, create one and make a note of it.
  - Call `OpenPort` to make the `grafPort` the current port.
  - If `StatusRecPtr` is `NIL`, use an internal one. This is to simplify your code; if the `StatusRecPtr` is `NIL`, you can reasonably choose not to use a status record at all, but this requires an extra code path.
  - Initialize the status record and the number of copies counter.
  - If the idle procedure pointer in the print record is `NIL`, point to an internal one. Again, as with the `StatusRecPtr`, you can choose to ignore idle procedures if no pointer is provided, but this requires an extra code path.
  - **Do The Following For Each Copy**:
    - Calculate the number of bands to print one page and initialize the page counter.
    - **Do The Following For Each Page:**
      - Call the idle procedure routine and initialize the band counter.
      - Get the handle to the picture associated with the current page.
      - Set the dirty flag in the status record to FALSE.
      - If manual paper feed, put up a dialog and wait for a response.
      - **Do The Following For Each Band:**
        - Call the idle procedure.
        - Calculate the band rectangle and update `iCurBand` with the current band number.
        - Call the idle procedure again.
        - Set the imaging flag in the status record to TRUE.
        - Call `InitPort` to reinitialize the port.
        - Adjust fields in the port to cause drawing into the band buffer.
        - Adjust fields in the location information field of the status record and calculate the sizing rectangle.
        - Calculate the boundary rectangle for the band and set the port rectangle to it.
        - Set the `ClipRgn` and the `VisRgn` to the sizing rectangle.
        - Initialize the band by filling it with white space.
        - Call `DrawPicture` to draw the picture into the band's rectangle.
        - Do whatever is needed to print the pixel image in the band's rectangle.
        - Clear the imaging flag.

        - Calculate the next band's position.
        - Increment the band's counter and loop back if not done.
    - If a vertical gap was specified, cause a form feed.
    - Increment the page count to the next page and loop back if not done.
  - Increment the number of copies counter and loop back if not done.
- Free any buffers that you own and close the port.
- Dispose of the information dialog that you put up.
- Dispose of each picture in the picture list by calling `KillPicture`.
- Dispose of the picture list itself.
- Restore the cursor.

Possible errors:

`portNotOn`	$1302	Indicates Port Not On
`prBozo`	$13FF	Someone unloaded the driver in the middle of the print loop

PrError ($1413)

Description:
Gets the error code from the last Print Manager call.

Passed:
None

Returned:

`LastError`	WORD	Result code from last Print Manager call

Performs the following:
- Gets the driver's internal error value (which was determined by the last driver call) and sets the return parameter `LastError` to it.

Possible Errors:

`noError`	$0000	
`PrAbort`	$0080	Indicates print job was aborted $1301 Indicates missing drivers $1302 Indicates Port Not On $1303 Indicates No Print Record $1306 Indicates PAP Connection Not Made $1307 Indicates Read/Write PAP Error $1308 Indicates Printer Connection Failed
`prBozo`	$13FF	Someone unloaded the driver in the middle of the print loop

PrSetError ($1513)

Description:
Sets the error value.

Passed:

`ErrorNumber`	WORD	Error number to be set

Returned:
None

Performs the following:
- Sets the driver's internal error value to the value of the passed `ErrorNumber` parameter.

GetDeviceName ($1713)
(also known as **PrChanged**)

Description:
Used as a communications tool between the printer driver and port driver.

Passed:
None

Returned:
None

Performs the following:

- Calls the port driver routine `PrDevPrChanged` with the printer name as input. This is necessary for drivers that work over AppleTalk. The name passed as the parameter to `PrDevPrChanged` should be what AppleTalk uses in an `NBPLookup` situation; for AppleTalk, such a name should follow Name Binding Protocol conventions.

  This routine will be called by the Print Manager when your driver is first loaded so a network port driver can find devices of your type. Applications should not make this call. When this routine will be called is not guaranteed; you can't use this as a substitute for a startup call.

PrDriverVer ($2313)

Description:
Returns the version number of the currently installed printer driver.

Passed:

`Wordspace`	WORD	Space for results

Returned:

`versionInfo`	WORD	Printer driver's version number

Performs the following:

- Gets the internal version number of the printer driver and returns it on the stack at `versionInfo`.

Note: The internal version number is stored major byte, minor byte (i.e., $0103 represents version 1.3)

PrGetPrinterSpecs ($1813)

Description:
Returns the type of printer and the printer's characteristics.

Passed:

`Wordspace`	WORD	Space for results

`Wordspace`	WORD	Space for results

Returned:

`PrinterType`	WORD	0 = undefined 1 = ImageWriter or ImageWriter II 2 = ImageWriter LQ 3 = LaserWriter family (except IISC) 4 = Epson $8001 = generic dot matrix printer $8003 = generic laser printer
`PrCharacteristics`	WORD	Bits 15 - 2 = reserved, must be zero Bits 1-0: 00 = cannot determine 01 = black and white only 10 = color capable

Performs the following:

- Returns characteristics intrinsic for the printer being supported.

The value returned for `PrinterType` should be the driver's `iDev` value.

PrGetPgOrientation ($3813)

Description:

Returns the page orientation from a print record.

Passed:

`Wordspace`	WORD	Space for result
`PrintRecordHandle`	LONG	Handle to the print record

Returned:

`PgOrientation`	WORD	Current page orientation: 0 = portrait 1 = landscape

Performs the following:

- Returns the page orientation from the current page setup information in the print record.

Immediate Mode Procedures

To print in the immediate mode, you need to install procedures which cause printing when you make QuickDraw II calls (as noted in `PrOpenPage`). This section describes the structure and parameters for these routines.

The basic idea is that your driver replaces low-level QuickDraw II routines with pointers to your own routines. For example, when someone wants QuickDraw II to draw some text (say with `DrawString`), QuickDraw II calls your low-level routine to draw the text. You can then print the text instead.

To install the immediate mode procedures, first create a procedure table for sixteen entries (16*4 bytes) and fill it with the standard procedures by calling `SetStdProcs`. Once you have the standard procedures, install the addresses of your replacement procedures into it and call `SetGrafProcs`. Installing your procedure addresses causes the appropriate QuickDraw II calls to call your procedures, which, in turn, perform the actual printing.

The routines that need to be written are known as QuickDraw II "bottleneck procedures." For most dot-matrix printer drivers, the one of most concern when writing immediate mode procedures is `StdText`. If your target device has an alternate page imaging language, you may wish to print entirely in immediate mode. In this case, you want to intercept most of the bottleneck procedures. Apple IIGS Technical Note #34, Low-Level QuickDraw II Routines, contains information on how to install these procedures. The sample code which follows shows how to replace `StdPixels` and `StdText`.

Example:

```
;****************************************************************
;** Example of Immediate Mode Printer Procedures.              **
;****************************************************************
Immedprocs     Start

SrcRect        equ $DC
SrcLocInfo     equ $CC
DrawVerb       equ $38
TextPtr        equ $da
TextLength     equ $d8
CharToDraw     equ $d6

;----------------------------------------------------------------
;
; StdPixels Procedure (Prints Pixel maps)
;
;----------------------------------------------------------------
Pixel          Entry

               phb                          ;save data bank reg on stack
               phk                          ;get program bank reg.
               plb                          ;use as data bank reg.

               lda iPrErr                   ;get errors
               beq Continue                 ;branch if none
               brl ExitPixel                ;branch if errors

Continue       anop

;This gets the source rectangle and stores it at PixelRect
               ldx #6
MoveSrc        lda SrcRect,x
               sta PixelRect,x
               dex
               dex
               bpl MoveSrc

;This gets the source LocInfo and stores it at PixelLoc
               ldx #16-2
MoveLI         lda SrcLocInfo,x
               sta PixelLoc,x
               dex
               dex
```

```
                bpl MoveLI

                pushlong #PixelLoc          ;push pointer to LocInfo
                pushlong #PixelRect         ;push pointer to rectangle
```

```
;++++++++++++++++++++++
; Insert code here to print a pixel map
;    INPUT:    PixelLoc   LONG, Pointer to pixel LocInfo
;              PixelRect  LONG, Pointer to pixels BoundsRect
;    SP->
;++++++++++++++++++++++

Exitpixel      lda #0                      ;return with no errors
               clc
               plb                         ;restore data bank
               rtl                         ;return with long

PixelLoc       ds 16                       ;pixel LocInfo
PixelRect      ds 8                        ;pixel rectangle

;---------------------------------------------------------------
;
; StdText Procedure (Prints Standard Text)
;
;---------------------------------------------------------------
StdText        Entry

               phb                         ;save data bank reg on stack
               phk                         ;get program bank reg.
               plb                         ;use as data bank reg.

               pushlong #PenPos
               _GetPen                     ;current pen pos. -> PenPos

;++++++++++++++++++++++
; Insert Code Here to move the printers head to the corresponding
; PenPos position (if needed).
;++++++++++++++++++++++

               pushword #0                 ;space for textwidth
                                           ;(for call to _TextWidth)

               lda DrawVerb                ;get DrawVerb
               beq DoCar                   ;if DrawVerb=0 then DoCar

               cmp #1
               beq Dotext2                 ;if DrawVerb=1 then Dotext2
;
;We get here if it's a "C" string (DrawVerb=2)
;
DoCstring      anop
               sep #$20
               longa off
;Search down through string looking for terminator to calc. length
               ldy #0
KeepLooking    lda [TextPtr],y
               beq TheEnd
               iny
               bra KeepLooking
TheEnd         rep #$20
               longa on
               lda TextPtr+2
               pha                         ;push the pointer to string
               lda Textptr
               pha
               phy                         ;push the length of sting
               bra Common
```

```
;
;We get here if it's just one character (DrawVerb=0)
;
DoCar           anop
                pushword #0
                tdc
                clc
                adc #CharToDraw             ;calculate addr. of char.
                pha                         ;push addr. of character
                pushword #1                 ;push length of one char.
                bra Common

;
;We get here if it's a string of text (DrawVerb=1)
;
DoText2         anop
                lda TextPtr+2
                pha                         ;push pointer to the string
                lda Textptr
                pha
                lda TextLength
                pha                         ;push the strings length
Common          lda 5,s                     ;Dup the last 3 words of
                pha                         ;the stack (for _TextWidth)
                lda 5,s
                pha
                lda 5,s
                pha
;++++++++++++++++++++++
; Insert code here to print the text
;
;    INPUT:     TextPointer    LONG, Pointer to text to print
;               TextLength     WORD, No. of bytes to print
;    SP->
;++++++++++++++++++++++
                _TextWidth                  ;get the texts width (DH)
                pushword #0                 ;set (DV)=0
                _Move                       ;move current pen location

ExitText        lda #0                      ;return with no errors
                clc
                plb                         ;restore data bank
                rtl                         ;returnith long

PenPos          ds 4                        ;pen position
                end
```

Further Reference

- *Apple IIGS Toolbox Reference*, Volumes 1–3
- Apple IIGS Technical Note #36, Port Driver Specifications
- Apple IIGS Technical Note #90, 65816 Tips and Pitfalls